AFRL-IF-RS-TR-2002-7
Final Technical Report
February 2002

AGENT-BASED CONFIGURABLE TESTBED

BBNT Solution LLC

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. J763

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

Copyright © 2000, 2001 by BBNT Solutions, LLC.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2002-7 has been reviewed and is approved for publication.

APPROVED:

DEBORAH A. CERINO
Project Engineer

FOR THE DIRECTOR:

MICHAEL TALBERT, Maj., USAF, Technical Advisor
Information Technology Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	FEBRUARY 2002	Final Mar 00 - Dec 01

4. TITLE AND SUBTITLE
AGENT-BASED CONFIGURABLE TESTBED

5. FUNDING NUMBERS
c - F30602-00-C-0082
PE - 63760E
PR - IAST
TA - 00
WU - 14

6. AUTHOR(S)
Richard Lazarus

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
BBNT Solutions LLC
10 Moulton Street
Cambridge MA 02138

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington Virginia 22203-1714

AFRL/IFTD
525 Brooks Road
Rome NY 13441-4505

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2002-7

11. SUPPLEMENTARY NOTES
Air Force Research Laboratory Project Engineer: Deborah Cerino/IFTD/(315) 330-1445

12a. DISTRIBUTION / AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 Words)***
The ABC Testbed Project spanned the period from April 2000 to December 2001. The project began as part of the DARPA Information Assurance and Survivability (IA&S) program and concluded as part of the DARPA Ultra*Log program. The project focus was to contribute experimentation methodologies and software tools for the understanding of composed, distributed systems with a particular emphasis on enhancing survivability of these systems. The idea was to capture the state of developing system and to visualize this system state in such a manner that it provides intuitive understanding of the system behavior. The ABC project culminated with the development of the Cougaar Society Monitoring and Analysis Reporting Tool (CSMART). Cougaar is a large-scale distributed agent application with minimal consideration for the underlying architecture infrastructure. CSMART is an integrated toolset for building, running, monitoring, and analyzing Cougaar societies, and for performing experiments on those societies by systematically carrying their properties and comparing the resulting behaviors.

14. SUBJECT TERMS
Distributed Agent-Based Systems, Survivable Distributed Agent Architecture, Logistics Planning and Support

15. NUMBER OF PAGES
52

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

C O N T E N T S

1. EXECUTIVE SUMMARY 1

1.1 PROJECT OVERVIEW 1
1.2 FINAL PRODUCT OVERVIEW 2

2. INTRODUCTION AND PROJECT HISTORY 4

2.1 INITIAL IASET GOALS 4
2.2 ULTRA*LOG GOALS 5
2.3 IMPLEMENTATION PHASES 5
2.4 ORGANIZATION OF THIS DOCUMENT 6

3. ABC TESTBED EXPERIMENTATION METHODOLOGY 7

4. ABC TESTBED DESCRIPTION 9

5. ABC TESTBED PLUGINS 11

5.1 ATTACKER AGENT PLUGINS 11
5.2 FIREWALL AGENT PLUGINS 12
5.3 VICTIM AGENT PLUGINS 13

6. OVERVIEW OF CSMART 14

6.1 CONCEPT OF OPERATIONS 14
6.2 CSMART EXPERIMENT COMPONENTS 15
6.2.1 Societies 15
6.2.2 Experiments 15
6.2.3 Impacts 16
6.2.4 Metrics 16
6.3 CSMART TOOL SUITE 16

7. THE ABC SOCIETY AND PLUGINS (BEHAVIOR MODELS) 18

7.1 ABC SOCIETY OVERVIEW 18
7.2 PLUGIN DETAILS 18

8. VISUALIZATION FOR ANALYZING SYSTEM BEHAVIOR 22

8.1 OVERVIEW 22
8.2 NAVIGATING THE VIEWS 23
8.3 COMMUNITY VIEW 24
8.4 AGENT VIEW 25
8.5 PLAN (EVENT) VIEW 25
8.5.1 Filtering 26
8.5.2 Legend 26
8.6 THREAD VIEW 27

8.7 GRAPH OBJECT DETAILS DIALOG....27
8.8 METRICS VIEWS....28

9. EXTERNAL EVENT MODELING....30

9.1 EFFECT-BASED MODELING CONCEPTS....30
9.2 THE TOOLS FOR MODELING EFFECTS....31
9.3 COMPONENTS OF THE EFFECT-MODELING PROCESS....31
9.3.1 Event Models, Classes, and Hierarchy....31
9.3.2 Impact Models....32
9.3.3 The Transducer....33
9.3.4 Infrastructure Hooks....35
9.4 DEVELOPMENT AND USAGE GUIDANCE....35
9.4.1 Current Extensions and Models....35
9.4.2 Creating new Event Types....41
9.5 INJECTING REAL WORLD EVENTS INTO COUGAAR....42
9.5.1 Editing Real World Events file....42

10. FUTURE DEVELOPMENT....44

11. REFERENCES....46

List of Figures

Figure 1: Example views Illustrating System Behavior 3
Figure 2: Validation Experiment Configuration 11
Figure 3: CSMART Tool Launcher, with Workspace Organizer 17
Figure 4: Society Monitor Tool Launcher 23
Figure 5: Navigation among different information displays 24
Figure 6: Community View for the ABC Society 24
Figure 7: Agents View, Highlighting a Relationship 25
Figure 8: Plan View, displaying the distributed blackboard 26
Figure 9: Thread View 27
Figure 10: Details Dialog, Plan view 28
Figure 11: Task Completion Chart 28
Figure 12: RealWorldEvent Hierarchy 32
Figure 13: ImpactModel Interface 33
Figure 14: Transducer Operation 33
Figure 15: Transduction Process 34
Figure 16: Intensity's Effect on Event Radius 39
Figure 17: Agent Down time 40
Figure 18: Node vs. Network recovery 41
Figure 19: RealWorldEvent DTD 42
Figure 20: Example RealWorldEvent list 43

List of Tables

Table 1: Cyber Attack Types 36
Table 2: Cyber Attack Event Fields 36
Table 3: CyberAttackEvents to Infrastructure Events 37
Table 4: SimpleKEvent Types 37
Table 5: SimpleKEvent Fields 37
Table 6: SimpleKEvent Type-characterization Variables 38
Table 7: SimpleKEvent Constraints 44

1. Executive Summary

1.1 Project Overview

The ABC Testbed project spanned the period from April 2000 to August 2001. The project began as part of the DARPA Information Assurance and Survivability (IA&S) program and concluded as part of the DARPA Ultra*Log program. The project focus was to contribute experimentation methodologies and software tools for the understanding of composed, distributed systems with a particular emphasis on enhancing the survivability of these systems. The idea behind the ABC Testbed effort was to capture the state of such systems and to visualize this system state in a manner that provides intuitive understanding of system behavior. This work was performed under Contract No. F30602-00-C-0082 with the Air Force Research Laboratory, Rome Research Site.

The ABC Testbed project culminated with the first delivery of the Cougaar Society Monitoring and Analysis Reporting Tool (CSMART). The CSMART product was published to the Cougaar open source web site for distribution to the Cougaar user community. CSMART enables Cougaar component developers and systems integrators to configure, execute, and analyze Cougaar agent societies with innovative visualization and automated configuration capabilities not previously available. Although CSMART was not the original intent of the ABC Testbed project, the ABC project contributed the basic foundations of CSMART. These foundations, especially exploration of system behavior through experimentation and user-driven visualization, were the initial pursuits of the ABC project and the ABC Testbed tool.

Cougaar is a large-scale agent architecture project that originated under DARPA, and has been made available to the general public through open source licensing. Cougaar provides developers with a framework to implement large-scale, distributed agent applications with minimal consideration for the underlying architecture and infrastructure. The Cougaar architecture uses the latest in agent-oriented, component-based design and has a long list of powerful features. Further, numerous leading edge technologies have been incorporated into the architecture that, we believe, puts this technology at the forefront of emerging agent architectures.

The DARPA Ultra*Log project is to extend Cougaar to become “survivable” while operating in chaotic wartime environments. The project is pursuing the development of technologies to enhance the security, robustness, and scalability of large-scale, distributed agent-based systems operating in adverse environments. The result of layering Ultra*Log technologies with the based Cougaar architecture will achieve a comprehensive capability that will enable a massive scale, trusted, distributed agent infrastructure for operational logistics to be survivable under the most extreme circumstances.

1.2 Final Product Overview

CSMART is an integrated toolset for building, running, monitoring, and analyzing regular Cougaar societies, and for performing experiments on those societies by systematically varying their properties and comparing the resulting behaviors. CSMART users will likely include Cougaar component/PlugIn developers as well as researchers interested in understanding Cougaar societies and mechanisms.

CSMART tools work within the Cougaar agent architecture to relieve developers and researchers from the myriad of details associated with editing configuration files, marshalling and controlling a set of host machines, distributing software, monitoring a running society, collecting data, invoking analysis applications, and identifying and saving the results. CSMART tools allow these users to concentrate instead on creating, measuring, and visualizing the behaviors of large-scale, agent systems while minimizing the complexities of specifying and constructing such agent societies.

The development of CSMART has focused on supporting the following activities:

- Configuration and control of abstract, regular Cougaar agent societies
- Performance monitoring of Cougaar agent societies
- Scalability and survivability analysis of Cougaar architecture/societies
- Providing an extensible base for adding new society specification, metrics, and monitoring tools

The CSMART methodology of performing experiments on simulated systems facilitates powerful research approaches. The most common and perhaps most powerful technique is to perform trade-off studies. Researchers can quickly re-execute experiments with slightly different initial conditions or with models of different fidelity. For survivability studies, researchers can perform multiple sets of trials, varying parameters of defense mechanisms, application configuration and topology, or types of attacks. The tools allow researchers to quickly define and run a set of trials; CSMART automatically collects the data, matched to the setup configuration, for later analysis.

CSMART includes a suite of visualization tools that provide developers and researchers insight into a society. Figure 1 illustrates several views that visualize the topology of the society, the relationships formed between agents during planning and execution, and the metrics generated by agents within a running society. Interfaces are included to transfer collected metrics to other specialized analysis tools for further investigation.
The need to accommodate large-scale societies requires that the visualization tools be capable of focusing and controlling the user's view. The tools can filter the data retrieved from a running society (for example, by focusing on selected communities or a restricted set of functional

relationships) and aggregate information for display to the user (for example, by abstracting the intermediary steps along a path). In addition, linkages between metric and topological views foster an incremental exploration of the society (for example, moving from metrics to related agents, or following threads formed by particular Task flows).

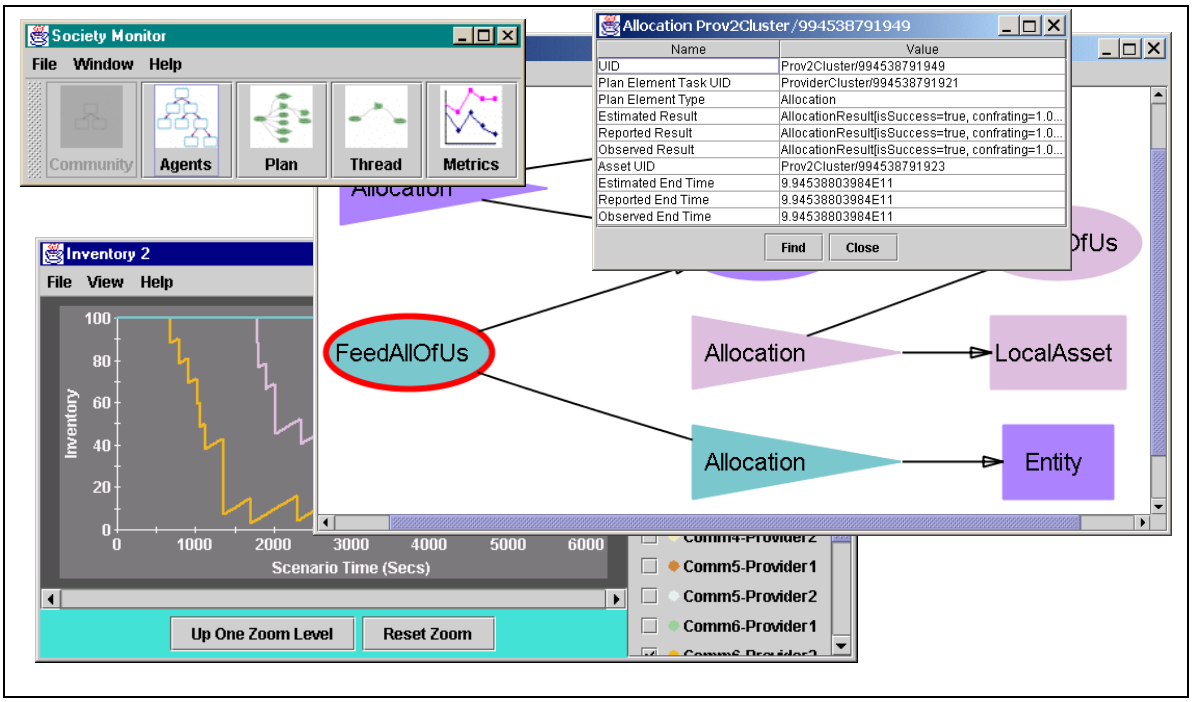


Figure 1: Example Views Illustrating System Behavior

2. Introduction and Project History

2.1 Initial IASET Goals

The ABC Testbed project began as part of the DARPA Information Assurance Science and Engineering Tools (IASET) program. IASET was one of six areas of the DARPA Information Assurance and Survivability (IA&S) program. The objective of the DARPA IASET program was the creation of science-based metrics, methodologies, and tools for the implementation of assurance in information system design and assessment processes. This program sought to overcome inherent deficiencies in the present processes in order to address assurance considerations with a true "system-level" viewpoint. Program achievements were to lead to more robust and secure systems that could be developed more rapidly, and have lower life cycle costs. ABC focused on developing a testbed for the implementation of assurance in information system design. In addition to developing the testbed, the ABC effort focused on validating the testbed's performance and usage methodology by developing IA component models and conducting experiments, as well as assisting others in conducting experiments.

The ABC Testbed and toolkit employed agent and component-based technologies to simulate and analyze IA behaviors. In this testbed, all entities of a data configuration are modeled as an independent agent whose detailed behaviors are modeled by a set of PlugIn modules. In such an environment, IA&S researchers could perform experimental analysis of different IA attack and defense strategies. Our goal was to demonstrate that the ABC Testbed provides not only a powerful and flexible tool for analyzing the security vulnerabilities of various configurations, but an efficient approach to the development and maintenance of such configurations.

The ABC Testbed intended to provide the following benefits:

- Analysis of complex interactions of multiple attacks and defenses, whether coordinated or independent, through the emergent behavior of composed entities
- Analysis of down-stream vulnerabilities triggered by induced "minor" vulnerabilities
- Allocation of model development across distributed groups with particular algorithms and capability expertise through a PlugIn component architecture
- Leverage existing and emerging IA&S models by capturing mixed-fidelity models in PlugIn modules
- Reduce experiment setup and integration time and costs by generating standard experimentation configurations and PlugIns that can be reused in multiple experiments
- Provide modeling and analysis of multiple hypotheses in a controlled experiment to measure the effectiveness of IA&S techniques in dynamic environments
- Provide precise specification of non-intrusive instrumentation

2.2 Ultra*Log Goals

With the reorganization of the DARPA Information Assurance and Survivability (IA&S) program and the abandonment of the IASET goals, the ABC Testbed project became part of the DARPA Ultra*Log program. Since the ABC Testbed was built on the Cougaar architecture, the Ultra*Log program was a natural fit as the ABC techniques for system experimentation and understanding could be focused on Cougaar system developers. As a result of this project redirection, the ABC Testbed was now focused on the creation of tools to stress and understand the behavior of Cougaar agent societies (rather than simulate arbitrary systems). In addition to developing these tools, the ABC Testbed attempted to validate tool performance and usage methodology by developing Cougaar component models and conducting experiments, as well as assisting other Ultra*Log developers in using our tools and conducting experiments.

The ABC Testbed and toolkit employed agent and component-based technologies (based on the Cougaar architecture) to simulate and analyze the behaviors of large-scale, composed systems with a focus on information assurance and survivability. As a part of the DARPA Ultra*Log program, the ABC project focused on:

- representing cyber attack events and physical disruptions of both consequential and malicious forms
- simulating the enormous complexity of an Ultra*Log system situated in a chaotic but realistic operational environment
- providing an environment in which to understand, measure and validate survivability techniques in an Ultra*Log system.

2.3 Implementation Phases

As a result of the ABC project redirection and its switch from the DARPA IASET program to the DARPA Ultra*Log program, the project design and implementation efforts were divided into two distinct phases.

- Phase 1 included the period of performance under the DARPA IASET program. During this period, the primary emphasis of the ABC project was on the simulation and exploration of distributed, heterogeneous systems with a focus on information assurance. To this end, the ABC Testbed employed the Cougaar agent architecture to develop a distributed simulation capability. In addition to the simulation testbed, the ABC effort developed an experimentation methodology and a set of powerful visualization tools for understanding the behavior of systems executed by the ABC Testbed. This phase culminated with the execution of a cyber attack on an information system emulated by the ABC Testbed.

- Phase 2 included the period of performance under the DARPA Ultra*Log program. During this period, the primary emphasis of the ABC project was on creating tools to automate the execution of Cougaar societies under stressful operation conditions. In addition, the project developed tools to capture and visualize Cougaar agent behavior to understand the impact of the operating conditions on these Cougaar agent societies. This

phase culminated with the initial release of the Cougaar Society Monitoring and Analysis Reporting Tool (CSMART).

2.4 Organization of This Document

Section 1 provides an Executive Summary of the project objectives and the final CSMART product. Section 2 (this section) provides a summary of the project goals, the transition from IASET to Ultra*Log, and the project development focus. Sections 3, 4 and 5 detail the ABC Testbed phase of the project including our experimentation approach and the ABC Testbed tool. Sections 6 and 7 focus on the usage and capabilities of our CSMART tool. Section 8 describes our visualization approach and products incorporated into the ABC Testbed and CSMART tools. Section 9 describes our external impact modeling approach and the capabilities developed for the ABC Testbed and CSMART tools. Section 10 concludes the report with a discussion of future work.

3. ABC Testbed Experimentation Methodology

The ABC Testbed provides a simulation and analysis environment, with an initial focus on the information assurance domain that combines modeling and experimentation. In this Testbed, independent agents, whose detailed behaviors are modeled by "PlugIn" modules, represent all aspects of system behavior. Each model includes the fidelity necessary to represent the behaviors of interest. Execution of these models produces a detailed, time-based Event Graph that captures both the course of actions executed by modeled system components and the evolving state of modeled system resources. For exploration and understanding of IA characteristics associated with these systems, the ABC Testbed provides tools for visualization and analysis of the Event Graph data.

The major output of an ABC experiment is a comprehensive Event Graph comprising all relevant actions of simulation entities and state histories of system resources. Example action events include requests for file access or requests to open/close a firewall port. Example state histories include the opening/closing of a firewall port or the escalation of policy. The Event Graph represents all relevant information for experiment comparison, for experiment control and "what-if" analysis, and for analysis tools that facilitate understanding system behavior.

In addition, the Event Graph explicitly represents the cause-and-effect relationships as computed by the behavioral models (PlugIns). PlugIns specify all the previous events that are causes for each event that they publish, as part of the event data structure. These are the "little whys," each of which is important. Together, these references and events comprise a causal network or graph of events. The combination of these individual explanations and the visualization of the "big picture" leads to an understanding of the security aspects of the system as a whole. This approach allows the researcher to make conclusions about the merits of the particular IA technique or component being evaluated.

As appropriate for a scalable distributed system, the data behind the Event Graph is distributed across the agent society. Each agent contains appropriate local segments with links to other agents that maintain cause and effect relationships. This information allows either dynamic or post-processing of these events to develop the metrics appropriate for understanding system behavior.

By executing models of suitable fidelity, we produce data that varies only according to the factors of interest, limiting the data and simplifying the analysis. By collecting comprehensive data on the actions and behavior of individual components, as well as the reasons for each individual action, we create a comprehensive collection of data on the behavior of the distributed systems, along with data on "why" the system behaves as it does. IA researchers can then investigate the workings of these systems and their IA or survivability attributes in particular, by analyzing the Event Graph.

While we recognize that models can never be accurate enough for formal validation of the security of a system, our focus is on directed research to understand system level behaviors and component interactions. One method of incorporating additional fidelity with minimal effort is to incorporate actual software components in place of more limited fidelity models. However, this diminishes the researcher's control over the experimental results. These issues are typical of any simulation-based analysis. Thus, we intend to complement live experimentation with ABC-based experimentation, not replace it.

In addition, modeling within the ABC Testbed can amplify the value of live experiments both before and after the live runs: modeling beforehand allows experimenters the luxury of dry-runs to ensure that the data that is being gathered can answer the questions that motivate the experiment. After an experiment has been run, data from the experiment can be used to tune the model, which can, in turn, be used to simulate many variations on the experimental situation, which variations were too expensive to try live.

4. ABC Testbed Description

The ABC Testbed employs a novel architectural approach to support distributed simulation. For our infrastructure, we use the open source Cougaar agent architecture. As we are focused on simulating large-scale, distributed systems, our infrastructure is an inherently distributed and scalable system. In addition, an agent-based architecture such as this provides a natural metaphor for modeling the behaviors of systems in a distributed manner. During the course of simulation, we can study the effects of the actions of the attack and defense mechanisms on the underlying system behavior. Because we are using a simulation environment, we can measure and log all aspects of performance in order to explore and understand the behavior of large, distributed, heterogeneous systems without compromising the performance of the modeled components.
The ABC Testbed utilizes a number of important Cougaar agent architectural features to enhance the simulation and investigation of the behavior of large-scale, distributed systems. The ABC Testbed builds upon the basic Cougaar architecture to provide the following capabilities:

A PlugIn-based component model: Within the ABC Testbed environment, all entities modeling behaviors or processes are considered "PlugIns." PlugIns encapsulate all domain aspects of the simulation, including modeled system behavior, wrapped actual software components (software in-the-loop), and human interactions (human in-the-loop). PlugIns are responsible for communicating their actions (as Events) as well as "why" they performed such actions. PlugIns adhere to a consistent API and object model that enables the composition of agents with arbitrary capabilities. A set of PlugIns comprises an agent and provides that agent with its particular behaviors and capabilities. For example, with a set of attack, telnet, detector, and firewall PlugIns, an investigator can construct experiments that analyze the deployment of distributed sensors or that analyze the effects of detector latency.

Distributed, virtual Event Graph: PlugIns interact with the local agent blackboard through a publish/subscribe mechanism. Each PlugIn subscribes to events of interest, and then responds to these events (or a pattern of events) by publishing a subsequent event (or pattern of events). ABC events are different from "external" events and they represent only the manifestation of "external" events within the components and assets of the simulation environment. PlugIns indicate the causal relationships (when possible) of all the events that they publish, as references to other events. Thus, the Event Graph (which represents the collection of all simulation information) forms a directed a-cyclic graph (DAG) and encompasses the collection of local agent blackboards across the society of agents. This Event Graph information can become large and possibly intractable for a single computer. As the agents represent "self-sufficient" compute entities, the agents can be distributed over an arbitrary number of computer platforms (one to a computer, at the limit) without incurring additional modeling work.

A composable object model: Our approach employs a hybrid class hierarchy and composable-object representation paradigm. In a composable object-based representation, one is concerned with the characteristics of an entity (what does it do) rather than the type of the entity (what it is). For example, the representation of software or applications as part of a system is difficult

(possibly intractable) as there are many types and variations on those types. Consider a firewall application. One instantiation might contain http and ftp services while another instantiation could contain ftp and ssh services. In addition, these firewall applications could be developed by different vendors and possess different properties. Using a composable object-based representation, one could enumerate an application class with a PortService property group or more specifically, a FTPPortService and a HTTPPortService. Another advantage of this approach is that the developer can dynamically instantiate objects with new or additional capabilities without recompiling the system – a great advantage in a world that is ill-defined and rapidly evolving.

Mixed-fidelity modeling support: To support the simulation and investigation of large-scale systems without incurring an immense modeling effort, our simulation construction and model approach specifically supports mixed-fidelity modeling. Our simulation paradigm comprises the modeling of a small set of high-fidelity components to support the specific investigation or hypothesis in conjunction with lower-fidelity models to provide the surrounding behavior of a large, distributed system. In this manner, actual software components can be incorporated into the simulation environment (as high-fidelity PlugIns).
While the Cougaar architecture provides us with an underlying agent infrastructure, it is an infrastructure that is optimized for performance with respect to the logistics domain, where the situation is constantly changing and time constants are relatively long. For this environment, the Cougaar architecture design is inherently asynchronous and non-deterministic – there is no consistency of asset state nor ordering of message delivery. While this is acceptable (preferable from a performance perspective) in the logistics domain, it is not acceptable for a simulation environment where repeatability is of prime importance. Thus, we re-implemented some of the core Cougaar components to achieve repeatable and deterministic behavior. These implementations include a new PlugIn base class (to abstract PlugIn mechanics from the modeler), object queue managers, and asset manager components to synchronize all PlugIn actions, asset state changes and message propagation.

5. ABC Testbed PlugIns

As we implemented the ABC Testbed system, we performed a series of experiments to validate our approach and implementation. For validation, we selected a previously conducted "live" experiment that employed a traditional red-team approach, and emulated this experiment within the ABC Testbed. By successfully modeling the significant system components and executing our models, we produced results consistent with those produced in the live experiment. In validation experiment, an attacker exploits a known weakness in the Pluggable Authentication Module of RedHat Linux 6.0 to gain root access to a system (the "pamslam" attack, CVE-2000-0052, published 1/4/2000). The attack used an automated attack tool, which was configured to make repeated attempts to gain root access until either it was successful or the number of attempts exceeded some limit beyond which visibility and risk of exposure was too great.

Figure 2 illustrates the experimental configuration. There were three hosts, modeled by three ABC agents: the Attacker, the Victim, and an intermediate Firewall. Each agent contained a few PlugIns, each modeling a specific software component executed during the AIA-001 experiment (e.g., telnet). The attacker script used telnet to access the Victim using a known account, built the attack script, and executed it. The detector on the Victim looked for the beginning of this sequence, waited the prescribed latency period, and then performed the appropriate response.

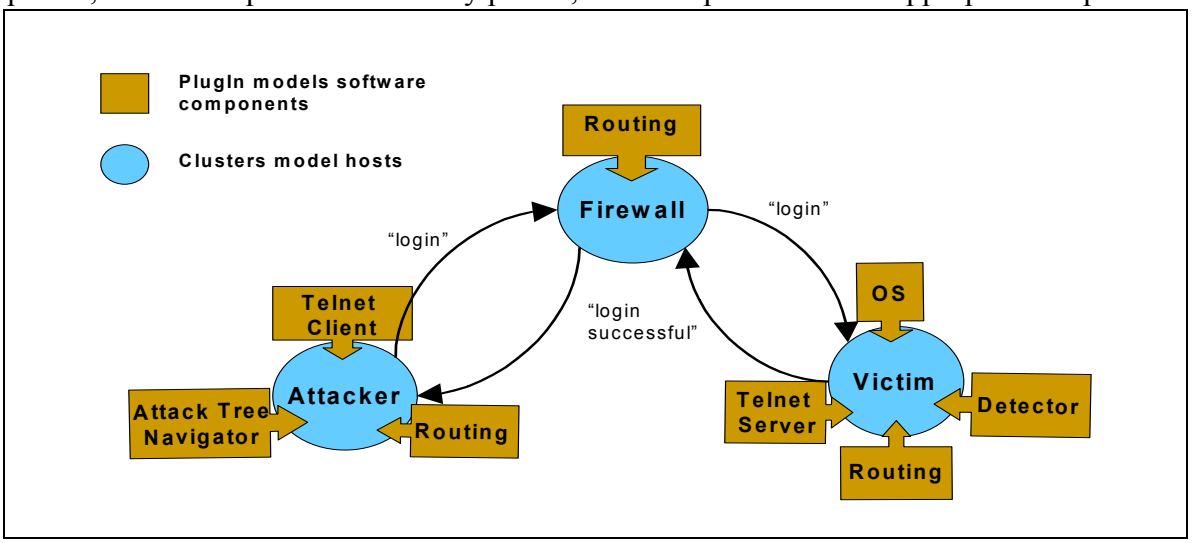


Figure 2: Validation Experiment Configuration

This experiment employed a number of PlugIns associated with the aforementioned agents (modeled hosts). These PlugIns are described below.

5.1 Attacker Agent PlugIns

The following PlugIns implement the behavior characteristics of the Attacker Agent.

- The Attack Tree Navigator PlugIn is responsible for modeling the attacker. This PlugIn will be the primary driver of the scenario. The PlugIn will load an attack tree into memory and navigate the tree to perpetrate the attack. The attack tree will contain all the data that will be necessary to form the commands (traffic events) that will be sent to the victim machine to gain root access. Since the AIA 001 experiment had a fairly simple attack tree (it was a single scripted attack) we will encode the necessary information in an XML file that will be interpreted at runtime to produce the attack tree. The Attack Tree Navigator will step through the states of the tree as it receives responses back from the victim host.
- As these attacks are perpetrated using the "telnet protocol," a telnet client will be modeled. The Telnet Client PlugIn will maintain the "session" with the victim host's "telnet server." As such it will wrap the remote commands and pass them along to the remote host via the routing PlugIn. This action can be thought of like the TCP/IP stack that adds and removes headers as packets are sent and received. This PlugIn will also be responsible for assigning session IDs to the outgoing sessions so the remote host can reference them for its remote work.
- The Routing PlugIn will be common to all the agents in this experiment. It is responsible for understanding how to "route" the modeled traffic from source to destination. In the Routing PlugIn, the routing table will be simple. If the destination is on its own "LAN," the Routing PlugIn will "address" the traffic to the destination host directly. If it is not, the traffic will be routed to the default gateway, which in this case is the firewall. For this experiment, topological adjacency will likely be configured with the agent's Routing PlugIn and loaded via a configuration file. Future versions will likely be able to determine adjacent hosts via some discovery mechanism, or address scheme.

5.2 Firewall Agent PlugIns

The following PlugIns implement the behavior characteristics of the Firewall Agent.

- Like the Routing PlugIn in the attacker host, the Routing PlugIn in the firewall will be responsible for routing traffic passed to it. The routing table will be static and loaded at run time. It describes the routing between the two networks that the firewall borders.
- The Detector PlugIn will look for anomalous behavior in the traffic and direct action by sending events to the Filter PlugIn (adding a new filter) or Session Manager PlugIn to kill an active session.
- The optional Filter PlugIn will provide a layering of traffic filtering based on a rule set. It will likely intercept traffic messages before they are handled by the Routing PlugIn. The actual interception may be handled coincidentally with the routing. The Routing PlugIn may change its routing plan based on data from this Filter PlugIn.
- The optional Session Manager PlugIn will manage the existing sessions being conducted through the firewall so that sessions can be killed at the firewall.

5.3 Victim Agent PlugIns

The following PlugIns implement the behavior characteristics of the Firewall Agent.

- The Telnet Server PlugIn will accept telnet events from the remote host and translate the contained commands into events for the OS PlugIn to consume. The Telnet server will be responsible for terminating the sessions either upon exit initiated by the remote host or by a forced termination by the detector.
- The OS PlugIn is responsible for accepting commands from the remote host and returning results. The OS PlugIn will also emulate file management functionality. As the attacker agent adds files, this PlugIn will update the file system state to reflect those editions. This PlugIn will also respond to queries for the contents of a directory (*e.g.*, an "ls" command) and commands to modify file state (*e.g.*, read/write permissions).
- The Detector PlugIn, will be looking for the anomalous events that indicate an intrusion. Like the detectors in the live experiment, these detectors will be specifically configured to detect the detectable events as they happen and react after the designated latency. The PlugIns will then direct the response to either kill a session or kill a session and remove the files placed on the host by the attacker. The Telnet Server PlugIn and OS PlugIn will execute these requests.
- The Routing PlugIn is as described in Section 5.1.

6. Overview of CSMART

With the association of the ABC project with Ultra*Log, the ABC project focused on the development of CSMART capabilities. The CSMART tool suite is organized in a framework that reflects a particular model of how they are used and how they fit together. Initial users include developers who are interested in testing, debugging, and validating their code. (Is the society configured properly? Is my PlugIn working? Do the agents behave as intended?) Other users include researchers interested in investigating society behaviors (What behaviors manifest in the society? How does it respond to external events? How does it respond to changes in the configuration of the society or to changes in the parameters associated with individual agents?)

6.1 Concept of Operations

The CSMART tool suite is designed to facilitate building, running, monitoring and analyzing a society. The framework uses many terms already familiar within the Cougaar community (society, host, node, agent, etc) and introduces some additional notions to help bind together the tools and their operation into coherent flow. A *workspace* is provided to organize configurations, resources, data, and other components of an investigation. It is modeled on a file browser with hierarchical folders and typed documents. CSMART's term for the object of the investigation is the *experiment*. An experiment is a collection of one or more *trials* related by systematically varying some aspects of the definition of a society or the environment in which the society is run. A trial is a fully specified (executable) and instrumented Cougaar society, and forms the basis for monitoring and data collection. Completed trials retain the definition of the society, the context in which it was run, and the data collected during the run.

CSMART provides a GUI for users to interact with visual objects that represent experiments, trials, societies, hosts, nodes, and other components. The GUI is organized around the following sequence of activities:

1) Build a society from a predefined template by specifying properties that determine its topology and the behavior of its agents and their PlugIns. For an experiment, some properties will be left unspecified at this stage.

2) Build an experiment around a society by specifying any remaining values. The set of properties and n umber of values for each property determine the number of trials in the experiment. An experiment also includes specifications for *impacts* and *metrics*. Impacts are external events—cyber attacks, kinetic events—that unfold while the society is running. Metrics refers to data collected while the society is running, that will be used for later analyses.

3) Install the experiment on a set of hosts by assigning agents to nodes and nodes to hosts, thereby committing actual computing resources to the experiment. Once this assignment is complete, start the experiment, which successively installs each trial configuration and starts the society. The status of each node (running, stopped, crashed), its time history of activity, and its transcript is available to monitor the progress of the experiment.

4) Inspect snapshots of the running society. Tools for interrogating the society allow the user to visually examine its organizational structure and the relationships that arise among its agents in the form of a directed graph whose "links" correspond to relationships and "nodes" to agents. Different views emphasize community structure, superior-subordinate relationships, individual Tasks, and planning decisions. These snapshots rely on information retrieved directly from the Cougaar blackboard.

5) Review data collected during a trial for later analysis. Once a trial has ended, and the society has stopped running, only data computed by its defined metrics are retained. The metric data are copied from the hosts and saved in the user's workspace automatically. Optionally, this data can be retrieved from the nodes while the society is running to monitor its behavior.

6) Compare trials by reviewing the saved metric data and any snapshots taken while the society was running. CSMART will maintain the association between the data and the definition for each trial, which includes the conditions under which it was run.

6.2 CSMART Experiment Components

CSMART uses several terms to describe the components which you build using the CSMART tools. Societies, Experiments, Impacts, and Metrics are the components that you build, configure, and manipulate using CSMART.

6.2.1 Societies

For CSMART purposes, a society is a collection of Cougaar agents, grouped into one or more communities. Societies are constructed from a template, and then parameterized to fully specify the desired communities, agents, and PlugIns. Currently, two society templates are included: an ABC society (developed under the ABC project effort) and a scalability society (developed under the ALP project effort). The Configuration Builder is used to tailor a society.

6.2.2 Experiments

To run a society, it must be included in an experiment. An experiment is made up of one or more trials. Trials are constructed by specifying the variation of experiment parameters, e.g., a PlugIn parameter, selected agents to run, impacts executed, the distribution of agents to nodes or nodes to hosts. For each trial, there may be some results – statistics about the run that are collected by one of the specified metrics.

6.2.3 Impacts

An impact is an external effect you wish to impose on a running Cougaar society. For example, loading the network with background traffic, killing nodes, telling the system some inventory has been lost, etc. These impacts might be real (really loading up the network), or simulated (via a Binder on the agent or the MessageTransport service). CSMART allows the user to specify that certain impacts should be included in a given experimental run.

ABC impacts are simulated cyber attacks on the cyber resources of the Cougaar society. They are specified by editing an XML file, and, then, specifying that file in the File Chooser when prompted.

6.2.4 Metrics

A metric is a set of run-time performance statistics about the running society, which you wish to collect. CSMART allows you to add one or more to your experiment. Examples include average and peak CPU utilization, total number of tasks in the agent blackboard, etc. Currently, both available societies always include a fixed set of metrics. The frequency of sampling however is controlled from within the Society Configuration tool.

6.3 CSMART Tool Suite

The CSMART Tool Launcher is the primary interface to CSMART, and integrates the various CSMART tools. From this interface, users create societies, experiments, metrics, and impacts. They then use the various tools to configure, run, monitor, and analyze Cougaar societies. The CSMART tools are displayed across the top of this window, as pictured in Figure. These tools are generally used in left-to-right order. The tools and their functions are:

- **Configuration Builder**: Edit society properties, such as the number of agents, or parameters for a given PlugIn.

- **Experiment Builder**: Put together the pieces of an experiment, by adding a society to it, and one or more impacts or metrics. In the future, this will be the tool for constructing multiple trials, varying parameters of your society, to understand some behavior in more detail. To run a society, you must include it in an experiment.

- **Experiment Controller**: Assign agents to nodes and nodes to hosts. Then, run, stop or abort the experiment, and display output from the experiment.

- **Society Monitor**: Examine a running society (or saved data from a previous run), by looking at the contents of its blackboard. Display graphs of agents and plan objects, and some simple metrics. This tool may be run stand-alone for use with non-CSMART Cougaar societies.

- **Performance Analyzer**: Display metric results from an experiment in progress, or a completed experiment.

For a comprehensive description of the CSMART tool suite, instructions on how to use the tool suite, and use case examples, please refer to the CSMART User's Guide.

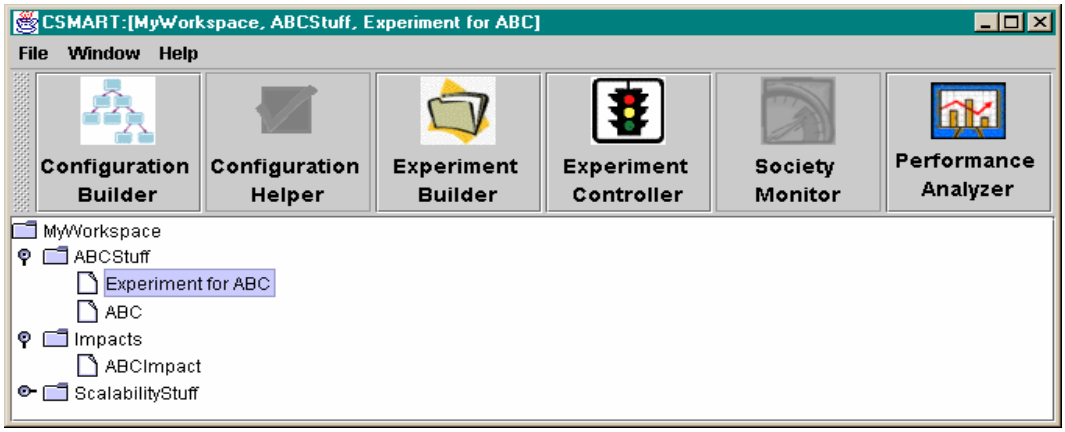


Figure 3: CSMART Tool Launcher, with Workspace Organizer

7. The ABC Society and PlugIns (Behavior Models)

7.1 ABC Society Overview

The ABC society was designed to model simple customer-to-provider request chains in a multi-agent society. In this society, the PlugIns use the planning language of Cougaar, such as Tasks and Allocations, but abstract away the complexities of the GLM module and existing higher-fidelity Ultra*Log PlugIns. A developer can use the ABC PlugIns to easily configure a large, complex multi-agent society with complex agent interactions. This ABC society can then be used to study agent support interactions and the effect of external impacts upon the society, such as degraded communication paths. Typical ABC societies are constructed in an "hourglass" topology. There are a number of customers injecting new demand (Tasks) into the system. All these requests funnel through some smaller number of distributors, and, then, out to a larger number of suppliers. These societies are representative of supply-chains and are particularly interesting to study in the context of adverse external impacts.

The society is organized by communities, which pass tasks between them. Each community comprises four agents. Two of these agents are Customers, which periodically inject new demand Tasks into the system. They are supplied by a Provider1 that has some local asset capacity. If that agent cannot fulfill the requests, they are allocated to the Provider2 within that community. If the Provider2 cannot fulfill the Task locally, it allocates the Task to the Provider2 in another community, depending how the supply relationships have been established. This society works in a purely execution mode: all tasks are matched against assets, and responses are always given with high confidence eventually. In particular, tasks include deadlines for completion, and often requests will fail due to the deadline being exceeded.

The primary PlugIns in the ABC society are Customers that generate new tasks, and Providers, which allocate these tasks to local assets or delegate them to other agents. The generation parameters and allocation rules are specified in customizable CSV (comma-separated-value) ".dat" files which are read by the PlugIns. By creating ".dat" files and agent ".ini" files the user can build large multi-agent societies without creating new Java-code or complex domain-specific PlugIns. An additional benefit of the ABC PlugIns is that they model basic reallocation and timeout-failures that model a robust society response to external impacts.

7.2 PlugIn Details

Most ABC PlugIns extend a "CSMART PlugIn" base class. This PlugIn provides some key support utilities to the ABC PlugIns, such as access to the CSMART "object factory" for creating ABC-specific data structures, logging support for detailed trace data, and utility methods for time-delayed operations and timers. An example of a time-delayed operation is "publishAddAt(Object o, long time)," which releases an Object to the agent Blackboard at a specified time in the future.

Four PlugIns comprise the modeled ABC domain in each agent:

- A **Local-Asset-Builder PlugIn** creates the local inventory assets using the CSMART object factory. These assets are created when the agent is first loaded, and their properties are specified in a ".dat" file. Currently the assets use a simple inventory design based upon a "thermal-resource-model." This model is conceptually similar to a well of water: the inventory has a maximum capacity that is gradually replenished over time, and every item consumed from the inventory has a fixed withdrawal quantity. The model additionally associates a usage-time for the used asset – all these parameters are specified in the ".dat" file and can be customized to model different inventory behaviors.

- A **Customer PlugIn** periodically creates new request tasks and injects them into its agent. A ".dat" file that specifies the task verbs and release frequencies configures the Customer. Task verbs are specified to have a particular verb (e.g. "Supply") and a deadline time by which this task must be either successfully allocated or failed. The Customer also maintains a "Happiness Chart" based upon the success/failure rate of its tasks, which is used by the CSMART metrics UI to summarize the Customer's interactions with the society.

- An **Allocator PlugIn** allocates the tasks to assets. A simple ".dat" file that configures an Allocator rule table also configures this PlugIn. The rule table maps task verbs to asset roles and is used by the Allocator to make its allocation decisions. When the Allocator receives a task it either allocates the task to a local inventory asset or delegates the task to a remote agent. An important function of the Allocator is to perform reallocations of failed tasks and to maintain "timeout" information for all Allocations – if an Allocation takes too long then the Allocator can cancel that Allocation and attempt to allocate to another asset or agent. This behavior makes the society far more robust to external impacts. Another responsibility for the Allocator is to propagate successful allocations upward to the task originator and create failed dispositions for tasks it is unable to allocate, either due to a task deadline or the exhaustion of its rule table.

- An **Executor PlugIn** handles allocations to a local inventory asset as created by the local Allocator PlugIn. The Executor manages the local assets created by the Local-Asset-Builder PlugIn. In particular, it examines the local inventory asset's capacity, denies allocations to empty inventories, and models usage-time for local assets (e.g. "Use of asset takes 10 seconds").

These four PlugIns do all the work in an ABC society, which makes the society behavior far easier to understand than a high-fidelity GLM/Ultra*Log society with many (100+) PlugIns and data structures. This simplicity will also allow us to extend these PlugIns to handle new Ultra*Log survivability requirements, such as agent mobility and QoS network information updates. As we require greater modeling sophistication, the ABC PlugIn suite can be enhanced and expanded to capture additional behavior requirements.

Three external-impact PlugIns model external attacks upon the ABC agents. Typically a society creator would create one agent to generate all the external attacks – this agent is called the "Generator" in the CSMART configuration builder. The second agent "transduces" the high level attacks specified by the user in the XML file, into the InfrastructureEvents that are sent to the individual agents. The three PlugIns responsible for this chain are:

- The **Scripted-Event PlugIn** loads the attacks and creates a high-level representation (see the Event Modeling section for details). This PlugIn reads an XML file and creates high-level CyberEvents and KineticEvents. An example of a CyberEvent is "Between 1pm and 2pm halt all Message I/O for agent X", and an example of a KineticEvent is "At 5pm bomb location Metropolis with a level-3 bomb". Associated with these high-level Events are models that define the low-level effects of the attack. The Scripted-Event PlugIn creates these Events.

- The **Transducer PlugIn** converts high-level attacks to low-level attacks. The Transducer listens for high-level Events generated by the Scripted-Event PlugIn and uses the attack models to create low-level InfrastructureEvents. For example, a KineticEvent "bomb" might translate into several Message I/O disruptions. The Transducer also maintains a model of the geographic location for all the agents, which is used to translate KineticEvent locations such as "15° Lat, 30° Long" to the agents in the vicinity of that location. Lastly the Transducer sends these InfrastructureEvents to all affected agents at the appropriate time.

- An **ABC-Impact PlugIn** is loaded into every agent in the society that you wish to impact, and performs the low-level attack. This PlugIn listens for InfrastructureEvents targeted at its agent and carries out the impact. To carry out message I/O impacts it uses the wrapped MessageTransport Controller described later in this document.

Together these three external-impact PlugIns create and coordinate a simulated attack upon an agent society, such as periodic network outages. A developer can use these PlugIns and the attack XML script to investigate the domain reaction to these attacks without actually toying with the network or worrying about complex machine setups.

Two infrastructure components assist the external impact implementation defined above:

- A **LogicProvider** (ImpactsLP) transfers InfrastructureEvents between agents. This Logic Provider is loaded by the CSMART domain, and runs like a PlugIn. Its simple job is to transfer InfrastructureEvents from the Transducer to the impacted agents.

- A **Binder** (SlowMessageTransportServiceFilter) assists the ABC Impact PlugIn. This Binder is loaded into every agent and wraps the MessageTransport. With the MessageTransport wrapped it keeps queues and timers to model degraded network performance, such as a network disconnection for 10 minutes. The wrapper also imposes a maximum throughput of input and output messages per second, which can be used to model a low-bandwidth agent. Lastly the Binder provides the ABC-Impact PlugIn with a

Controller service (SlowMessageTransportServiceProxyController) that allows the ABC-Impact PlugIn to carry out I/O performance degradations.

8. Visualization for Analyzing System Behavior

8.1 Overview

Exploration and understanding survivability mechanisms for large-scale, distributed systems is a daunting task. Even capturing and visualizing behavior of such systems that are functioning normally is a challenging problem due to its nature (large, distributed and composed). These difficulties are compounded when assessing the performance of multiple security and recovery mechanisms that comprise the survivability architecture for a large-scale, distributed systems. This occurs because these components tend to interact in both intended and unintended ways. A firewall may block the data that a detector needs in order to diagnose an attack. A trust-based monitor may shut down a component and thereby cause an abnormal traffic flow. The IA effectiveness of such systems depend on the survivability components interacting in the way that they were intended and not in destructive ways, yet the interactions among these components are complex and often cannot be predicted by any tractable method. The success or failure of the survivability system hinges on the emergent behavior of its components acting in concert. In general, the only way to discover this emergent behavior is to execute the software under various conditions, record what transpires, and analyze the resulting behavior.

To this end, the ABC project has focused on scalable approaches for collecting information and visualization tools for guiding and aiding the user in understanding the behavior of large-scale, distributed systems. As part of this approach, we have focused on representing the behavior of such systems as directed graphs and on providing the visualization tools to explore, compare, and analyze this information. The directed graph explicitly represents the cause-and-effect relationships as computed by the behavioral models and compute engines (i.e., PlugIns). For the ABC Testbed, PlugIns modeled or encapsulated system behaviors and published all their actions as causal events, as well as the reasons for taking these actions (as links to preceding events). For Cougaar logistics societies, the workflow and task objects form a causal, directed graph of system actions. In this manner, these PlugIns specify all the previous actions that are causes for each action that they publish, as part of the graph data structure. These are the "little whys," each of which is important. Together, these references and events comprise a causal network representing the system behaviors. The combination of these individual explanations and the visualization of the "big picture" leads to an understanding of the individual system behaviors, as well as for the system as a whole.

As appropriate for a scalable, distributed system, the data behind the directed graph is distributed across the agent society. For large-scale distributed systems, no one computer can compute or maintain the magnitude of this data. Thus, each agent contains appropriate local segments with links to other agents that maintain cause and effect relationships. These visualization tools appropriately retrieve the particular information of interest as express by user gestures.

Within CSMART, the Society Monitor tool displays the activities and data associated with a distributed running Cougaar society. Figure 4 illustrates the view launcher for the Society Monitor visualization tools. This data is obtained by collecting information from the contents of each agent's Blackboard, or from the results of specific metrics collection plugins (or by loading saved graph or metrics data). This tool allows users to monitor any currently running Cougaar society.

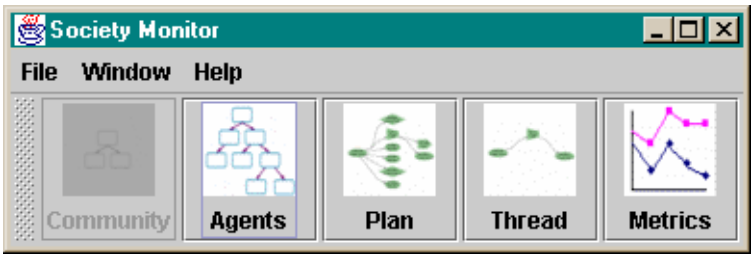


Figure 4: Society Monitor Tool Launcher

Views within this tool communicate via the standard PSP interface with a running society. When running under CSMART, the Society Monitor is enabled only when an experiment is running, and automatically contacts the society in the running experiment. When running standalone, the user is prompted for a URL in the running society. For developing the graph visualization tools, we have built upon the open source AT&T GraphViz package, as well as contributed additional functionality to this open source product.

The Society Monitor currently provides the following displays:

- **Community View**: The communities in the society and their relationships.
- **Agent View**: The agents in the society and their relationships to each other.
- **Plan (Event) View**: A directed graph of plan objects or events generated by the agent society.
- **Thread View**: Drill down across the whole graph only displaying objects related to a single selected object.
- **Metrics Views**: Show one or more calculated metrics as chart displays.

8.2 Navigating the Views

A researcher analyzing results produced by CSMART or the ABC Testbed begins with a high-level summary of system behavior, e.g., a "quality of service" display or an agent/host metrics display. From these summary views, the researcher identifies regions of interest, and navigates to the appropriate regions of the graph view. Figure 5 illustrates this navigation process, showing how a user navigates from a tabular display of derived data (View A), to a temporal plot of a single variable (View B), to a display of the subset of events occurring during an observed anomaly (View C), where View C is the Plan (Graph) View.

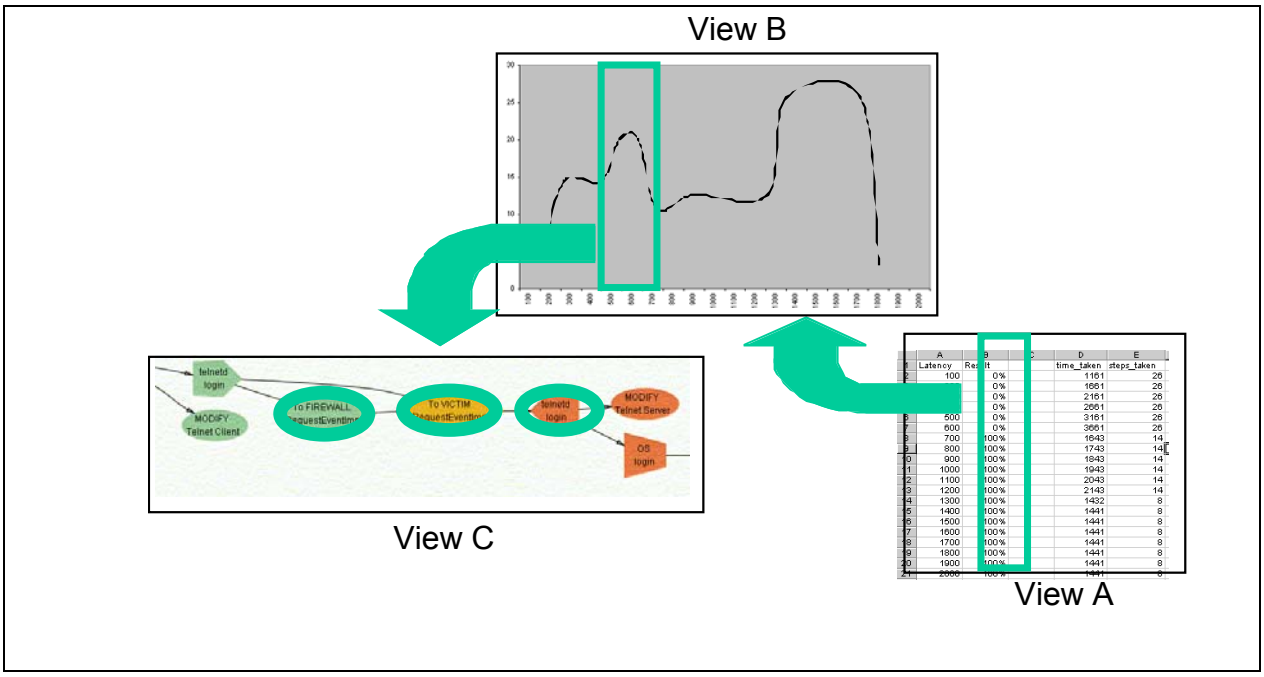


Figure 5: Navigation among different information displays

8.3 Community View

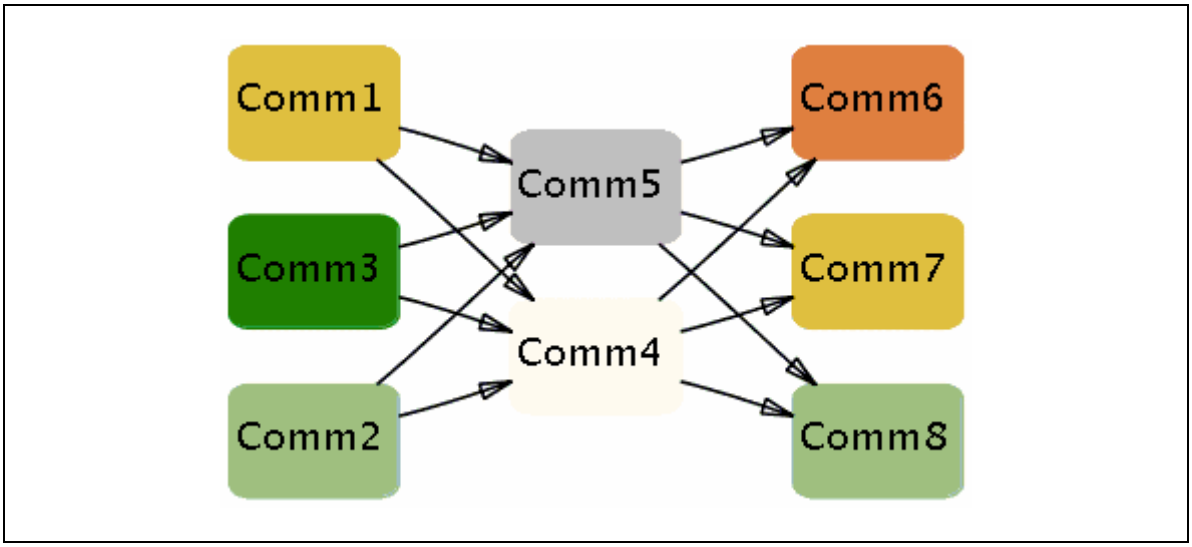
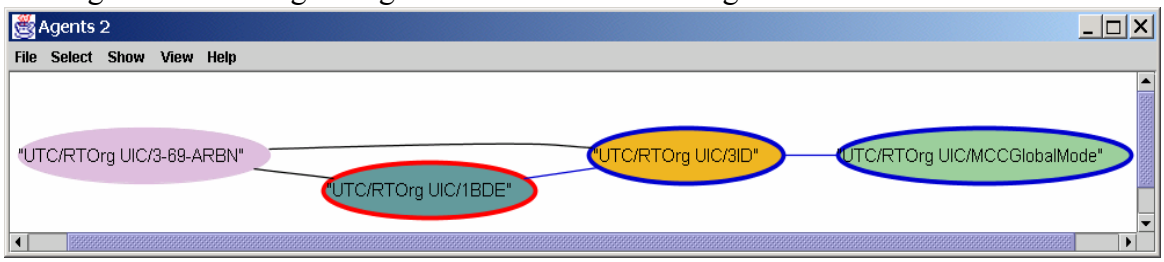


Figure 6: Community View for the ABC Society

- The Community View illustrates the communities in a running society with their relationships (the sum of the relationships of their agents). Figure 6 illustrates a sample Community View from the ABC society (each community comprises 4 agents). The Community View is the highest-level summary graph of the information flow across entities in the agent society. In societies of 100s of agents, the Community View presents a view of the entire society with greatly reduced clutter that can be useful in identifying communication flow patterns and potential bottlenecks.

8.4 Agent View

The Society Monitor Agent View displays the agents in a running Cougaar society. Figure 7 illustrates an example of the Agent View. Each agent is labeled and color coded. Each agent is linked to every other agent with which it has a relationship. A user may produce new graphs showing only selected relationships or only the relationships for a particular agent. In addition to the graphical attributes of an agent (e.g., name and relationships), a user can select a particular agent and generate a dialog listing all the attributes of that agent.



- Figure 7: Agents View, Highlighting a Relationship

8.5 Plan (Event) View

The Plan View is the most detailed monitoring tool available. It illustrates the information from the agent Blackboards (i.e., plan elements) as a causal (directed) graph. It can potentially return every object from the Blackboards of all the distributed agents, and illustrate the relationships among these objects. These plan elements represent the actions or behavior of the agents. For understanding a particular interaction or result, this is the most powerful and detailed view of the agent society behavior.

This view contains a number of features to help the user manage the complexity of Plan View, which can grow to be very large (1000s of objects). Multiple windows can be opened on the same graph to inspect disjoint portions of the graph. Subgraphs can be selected and copied to other windows; and selection and highlighting are reflected in all windows showing the same events. An overview window (bird's eye view) shows the entire graph and provides a reference frame for creating and identifying other views. Figure 8 illustrates a sample Plan View where objects were collected from multiple agents (the objects are color coded by agent). Note that the Plan View is laid out left to right, following the arrows from Tasks to Plan Elements to expanded

Tasks. These links are causal, and are drawn to minimize link length and crossing (thus, the layout does not reflect chronological order).
Manual interaction is only the first step in making the Plan View tractable as an analysis tool. We have experimented with forms of filtering and transforming graphs to simplify the task of detecting features in the causal relationships among plan elements.

8.5.1 Filtering

Typically, this view returns too much information (both to be useful, and to be handled comfortably by the client machine). Therefore, users typically filter their query: by community, agent, object type, some attributes of the objects, a time span, or other filter criteria.
Currently, users may choose to hide or ignore certain plan object types. Note that by "Hide"-ing an object type, it is still retrieved and processed, merely not drawn. The transitive closure of the links between nodes is drawn, with the edge shown thicker and darker as a reminder of the missing nodes. When ignoring objects, they are not retrieved from the agent at all. Therefore, for efficiency purposes, ignoring some object types is usually more useful.
Additionally, users may limit their query to some number of objects returned. Note that the set of objects that is returned is arbitrary. Look at the output in the console for some indication of how many objects were not returned.
In future, this view will support much more complex methods for filtering and finding, with complex pattern matching and graph abstraction techniques.

8.5.2 Legend

Nodes in the Plan View are color coded, according to the agent from which they were retrieved. Although this means that colors may be re-used, it does allow the user to quickly identify when processing moves from one agent to another. For details on which color indicates which agent, view the legend (from the "View" menu). Nodes are also different shapes, to indicate different object types, as defined in the legend.

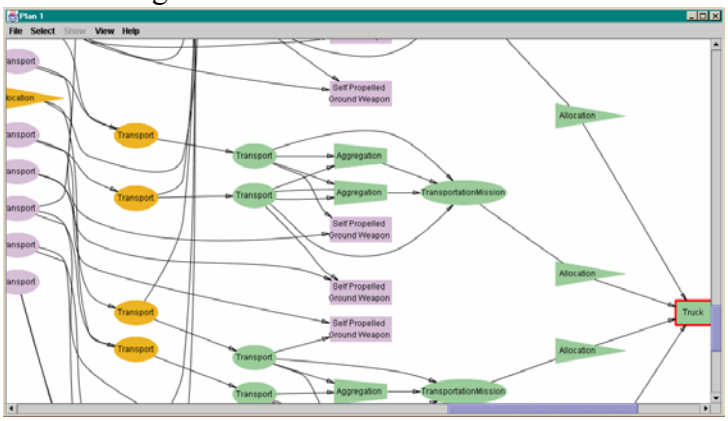


Figure 8: Plan View, displaying the distributed blackboard

8.6 Thread View

The Thread View is a filtered version of the Plan View. By selecting a particular node (usually from a Plan View), you may query up ("Ancestor") or down ("Descendant") its causal chain. This produces a new Plan Graph, with only the specified objects. When viewing an ancestor thread, only parent Tasks are displayed. When viewing a descendant thread, all objects that can be reached by following links are displayed. Figure 9 illustrates a sample Thread View that depicts all the "descendants" of the "Supply500MREs" task.

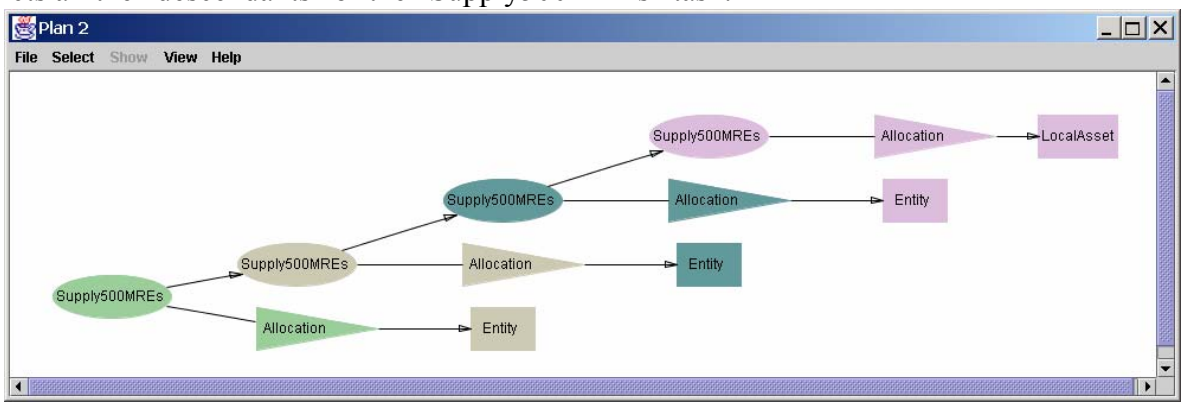


Figure 9: Thread View

The Thread View can answer the following questions:

- How was this Task satisfied?
- Who participated in the solution for this Task?
- What are the requirements associated with this Task?
- Who generated the requirements for this Task?

8.7 Graph Object Details Dialog

For each node in one of the graph views (e.g., Agent View and Plan View), double-clicking on the node retrieves a set of attributes for that object. These are a sub-set of the attributes available from the Blackboard. In future, this set will be tunable. Note that the "Find" button on this window will scroll the main graph to the appropriate node (the node corresponding to the Details Dialog). Figure 10 depicts a sample Details Dialog for a plan element from a Plan View graph.

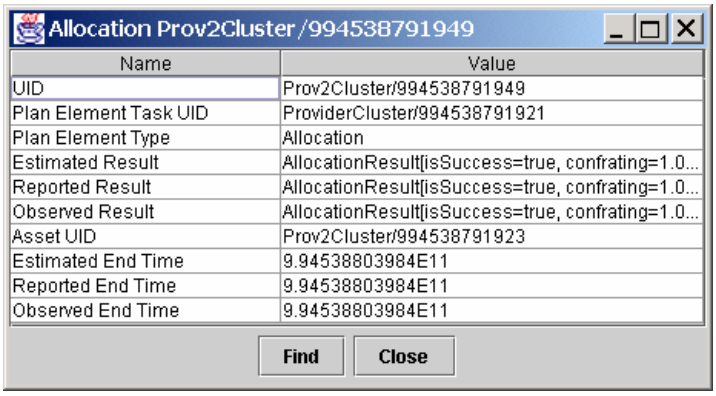


Figure 10: Details Dialog, Plan View

8.8 Metrics Views

Various run-time metrics may also be displayed from the Society Monitor. The Task Completion chart is a stacked, bar chart that displays information about Tasks for particular agents within the society. The displayed Task state is the society state at the moment the metrics button is pressed. Figure 11 illustrates a sample Task Completion chart.

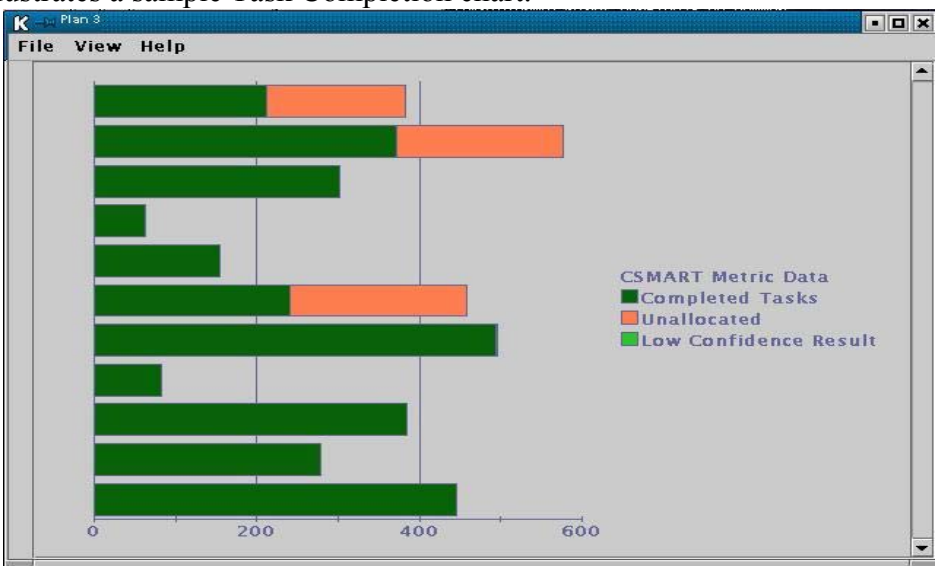


Figure 11: Task Completion Chart

The bar chart displays a bar for each agent within the society. Each bar consists of 3 sections:

- Completed Tasks – This is the total number of Tasks in the agent that have a high-confidence Allocation Result.
- Total Unallocated Tasks – This is the total number of Tasks that do not have an associated Plan Element within the agent.

- Total Low Confidence Tasks – This is the total number of Tasks with a low confidence (< 50%) AllocationResult within the agent.

At any point, the total bar length for each agent displays the total number of Tasks within that agent. Note that bars may shrink during a society run, due to Tasks being rescinded.

9. External Event Modeling

A key contribution of the ABC project was an investigation into modeling adverse real world events and implementation mechanisms by which to stress the modeled system for analysis. In the context of Cougaar society experiments, we define *events* as abnormal occurrences outside of the agent system that affect its operation. In the context of information assurance experiments, we define *events* as abnormal occurrences outside of the modeled system that affect its operation. Examples might be explosions, failures of supporting equipment, intentional system reconfigurations, cyber attacks, and unusual workload. Rather than attempting to produce an ontology or catalog of all possible event types, we have developed a systematic method for characterizing events in terms of their *effects* on a running society. This approach characterizes events in terms of their basic behavior rather than their type. In fact, we have found that the set of basic behavioral effects on a society is much smaller and much more manageable than the set of all possible event types.

One benefit to running Cougaar experiments with CSMART (or the ABC Testbed) is to evaluate system performance under stress or under adverse operating conditions. Therefore, we have attempted to provide a method for defining events that focuses on their effects on the agent system. We have limited our current work to modeling the cyber-effects of external events, that is, their effect on the software and hardware of the system under test. While cyber-events (attacks, system faults etc.) will have only cyber-effects, other kinds of events, such as kinetic events, might have significant indirect impacts on the agent society by affecting the outside world. These external effects might take two paths; effects on the operators of the system and effects on the world that the system is monitoring and controlling (such as kinetic events that have impact on the transportation infrastructure). This later sort of effect can produce system loads that, in turn, may impact the system's robustness.

In following this approach, we have developed an approach for describing events and tools for emulating their cyber-effects in a straightforward, uniform way that offers promise for modeling many events through the use of a few simple effect primitives.

9.1 Effect-based Modeling Concepts

In following this effect-centered approach to modeling events, we have applied the following concepts:

- *Event* – An occurrence beyond the normal operation of a system, such as a power outage, that will have an effect on a system. Also called a *real-world event* or an *external event*.

- *Event Model* – A description of a real-world event. An event model produces an *impact* that it sends to a *transducer*.

- *Event Class* – A limited number of broad, effect-based categories under which event models can be placed.

- *Transducer* – A component of the simulation (typically implemented as a PlugIn) that receives an impact from an event model and produces appropriate *society-specific impacts* that it applies to the system.

    - *Effect* – The response of a system to the specific impact of an event. Sometimes called an *infrastructure event*.

    - *Agent* – A course-grained component of the system that embodies system behavior.

    - *Infrastructure Hook* – The aspects of an agent that receives a specific impact and produces an effect based on that impact. These "hooks" are transparent to the agent components such that the effects emulate naturally occurring system state. These "hooks" are typically implemented as "binders."

9.2 The Tools for Modeling Effects

CSMART (or the ABC Testbed) contains a set of tools that allows experimenters to define events in terms of their effects. Using these tools, experimenters can devise experiments that model a system's reaction to those events. These tools include:

- A list of the important aspects of events that need to be specified in order to model that type of event in an Ultra*Log simulation.

- A language for describing low-level impacts on modeled components.

- A language for describing how events can be expanded into impacts.

- Tools that support experimenters as they build and run system and event models.

9.3 Components of the Effect-modeling Process

9.3.1 Event Models, Classes, and Hierarchy

In our effort to define an ontology of events to classify real world activities, we have implemented three classes of events: cyber attacks, kinetic events, and chaos (we have defined chaos as a way to introduce colored noise models into the system). Of course, we expect this ontology to evolve with additional effort and participation. These classifications are based on the type of impact the event produces. For example a moving event (e.g., a hurricane) would have a different impact model than a non-moving event (e.g., an explosion). Likewise, a cyber event would have a different impact model than a kinetic event.

We have defined models for each class of event, or for some abstraction level of these events. These parameterized models define the impacts of the external event in terms of time course, geographic course, intensity, etc. We intend to provide a generic set of these models that IA and Cougaar developers can extend. We are currently defining generic models for each event class.

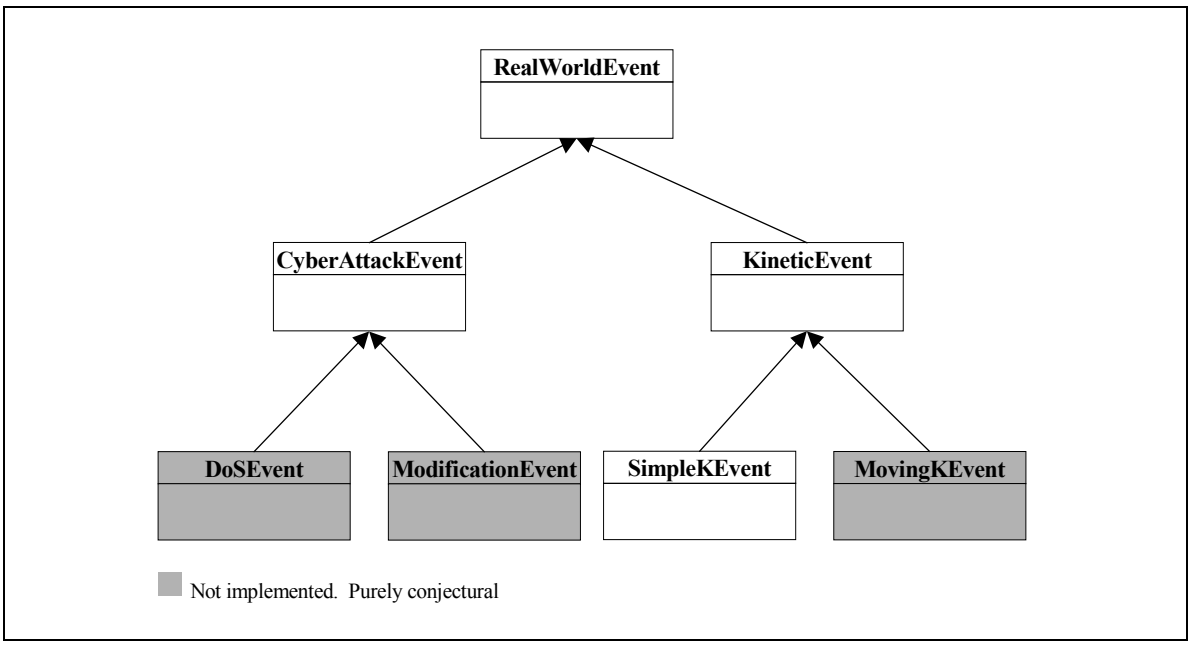


Figure 12: RealWorldEvent Hierarchy

Figure 12 gives a simple event hierarchy. This hierarchy is expanded beyond our current implementation to give a better understanding of the direction we are heading. The details of these classes are given below.

9.3.2 Impact Models

Impact models are objects that simulate the impact of an event. Impact model objects implement the ImpactModel interface (shown in Figure 13). Currently the classes are implemented as inner classes to the real world event that utilizes them. Alternatively an entire ImpactModel class hierarchy could be developed. The getImpact() method should return a collection (as an Iterator) of unpublished InfrastructureEvents. These InfrastructureEvents should be made ready to publish within the getImpact() method. This means that visibility time, causes and all other data members should set, leaving only the job of publishing to the transducer. The cause of the resulting InfrastructureEvents should be the `RealWorldEvent` itself.

```
Package org.cougaar.tools.csmart.ldm.event;
import java.util.Iterator;
import org.cougaar.tools.csmart.ldm.PlugIn.transducer.Society;
public interface ImpactModel {
    public Iterator getImpact(Society world, IEFactory theIEF);
}
```

Figure 13: ImpactModel Interface

9.3.3 The Transducer

This component maps a description of an external event into a set of specific impacts that act upon individual agents in the modeled society. The transducer uses knowledge of the society, such as the geographic location of the agents and knowledge of the event types, to produce appropriate specific impacts at the right times.

The transducer determines the state of the society and maintains it. When the transducer receives a real world event it gets the impact model from the event and calls the getImpact(society) method on the model with the society object, which is a single object representing the society. This society state need only contain information relevant to making the translation. Thus some coordination is required to ensure the proper state is maintained for the collection of real world events that will be handled. A simple Society object is now in place to handle the simple events we have implemented. In the future more complex real world events may require modification to this object.

It is the responsibility of the transducer to understand the society enough to execute this translation. For instance, an explosion whose location is given by a lat-long pair must be translated to the Cougaar agents that are within some distance of the explosion's epicenter (as illustrated in Figure 14). The transducer in this case must be able to lookup agents based on lat-long pairs.

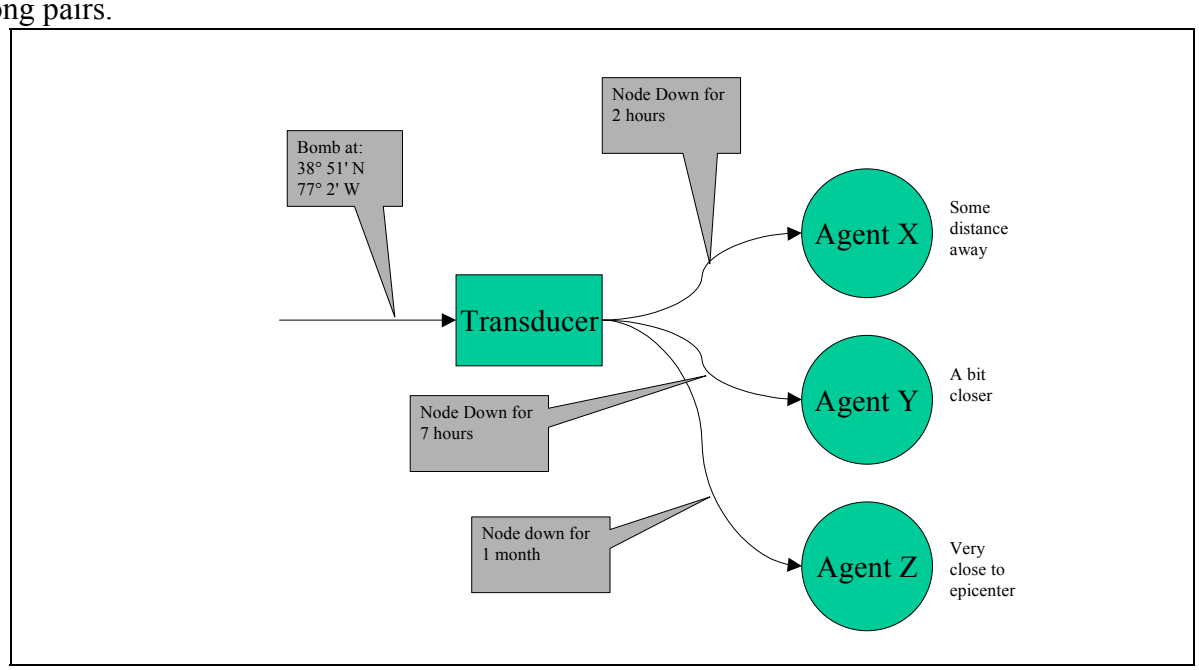


Figure 14: Transducer Operation

Decomposition is necessary on many levels depending on the type of real world event. It is conceivable that a cyber event, such as a data trunk being severed, will need to be translated into degradation of communication service between certain nodes. The transducer in this case would have to understand the topology and routing used by various nodes to determine the correct impact.

Thus, the transducer component must:

- Understand the state of the society for which it is transducing.
- Translate the real world events into infrastructure events using the society state. This translation is very much dependent on the type of real world event perpetrated.

As new event classes are added, the transduction process will get complex, as various event classes will have to be handled differently based upon class member values, or the state of the society. To handle this, we divided responsibility for the transduction process as follows:

- The transducer will be responsible for maintaining society state.
- The impact model will contain the algorithm for translation of a particular event impact.

Figure 15 illustrates this transduction process. The transducer receives the impact model from a method on the event model. As such, the event model contains all the algorithms for translation, lacking only the state of the system, which it obtains from the transducer.

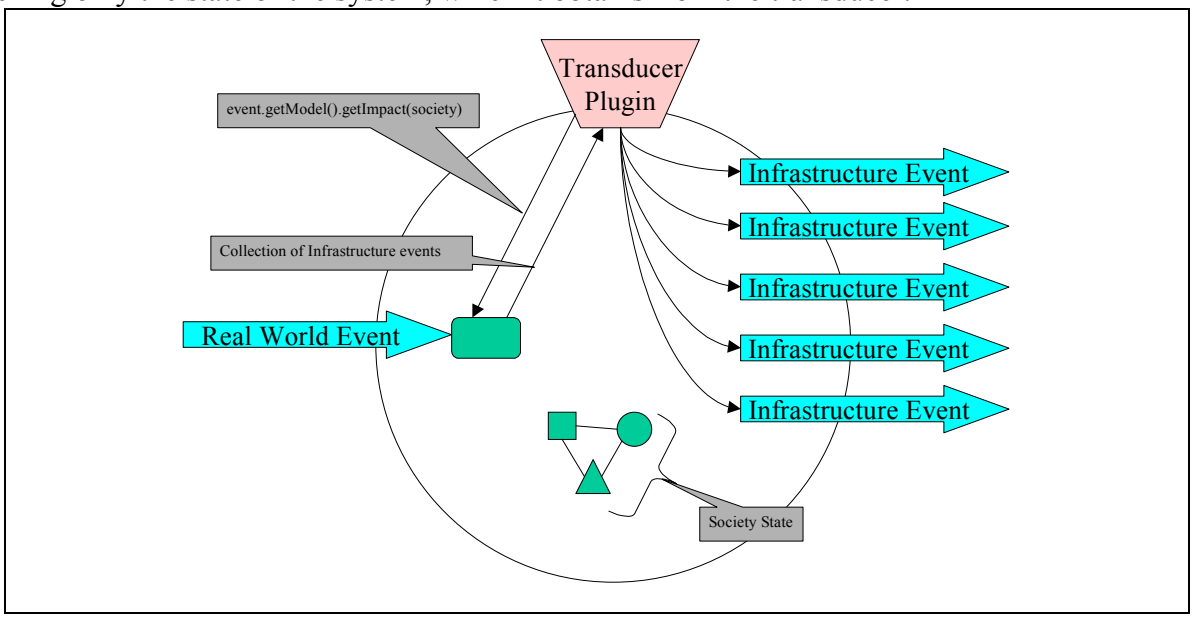


Figure 15: Transduction Process

9.3.4 Infrastructure Hooks

Infrastructure hooks allow modeled agents to respond to the specific impacts generated by the transducer. These hooks implement effects such as the reduction of CPU bandwidth or the isolation of specific agents from the network.
The CSMART infrastructure supports infrastructure attacks and impacts that:

- Degrade or halt the processing speed ("node") of an agent for a specified duration
- Degrade or halt the network connectivity ("wire") of an agent for a specified duration

The intensity of an impact can be between 0% (no impact) to 100% (halt). Impacts are targeted to a single agent and have a constant intensity for their specified duration. Impacts are additive, such that multiple external events or attacks occurring at the same time have a greater cumulative impact than a single event.

These infrastructure hooks are sufficient for emulating system-operating conditions for most PlugIn binders and PlugIns. While these impacts are rather limited, they are able to emulate the impacts of a wide range of malicious and kinetic attacks. As Cougaar survivability mechanisms become more sophisticated, our implementations will have to accommodate emulating these impacts in a manner such that they present standard information flow to Cougaar components and survivability mechanisms.

Note: These infrastructure hooks will be available as a Cougaar architecture test mechanism and are not intended to be available in the deployed Cougaar system.

9.4 Development and Usage Guidance

Developers can contribute in a number of ways to the enhancement of real world event generation and execution. Primarily, developers can extend the current set of Real World event types to allow for a greater breadth of capabilities. Developers can also develop new algorithms (in the form of PlugIns) for the creation of real world events, such as smart attackers and weather simulators. This section provides guidance to developers on the extension of real world events.

9.4.1 Current Extensions and Models

To aid the developer, below is an overview of the real world event classes that we have currently implemented.

9.4.1.1 CyberAttackEvent

CyberAttackEvents are used to describe some nefarious action on behalf of some modeled cyber attacker. At the highest level, this may include denial of service attacks, modification attacks, destructive attacks, etc. As an initial simplification, we have focused on denial of service or degradation of service attacks. Later, this type of attack may exist as a subclass of

CyberAttackEvent, but currently does not. There are four types of cyber attack and those are listed in org.cougaar.tools.csmart.Constants.RWEType and described in Table 1.

Table 1: Cyber Attack Types

Constant	*Value*	*Attack Description*
DOSNODE	"Dos Node"	Launch a denial of service against a single agent. Will result in a degradation of service of the entire node.
DOSNET	"Dos Net"	Launch a denial of service against a network. This will result in a degradation of Inter-agent communication (message passing). Currently we assume there is no more than one node per network. In fact this is directed against an agent and should be interpreted as "launch a DoS against the network this agent is connected to."
KILLNODE	"Kill Node"	Bring down a host. This will result in a termination of all node activity for a period of time.
ISOLATENODE	"Isolate Node"	Bring down the network an agent is connected to. This will result in a termination of communication to and from the agent in question.

The fields of a CyberAttackEvent are shown in Table 2.

Table 2: Cyber Attack Event Fields

Field Name	*Possible Values*	*Description*
Type (from RealWorldEvent)	Constant from Table 1**Error! Reference source not found.**	The type of Real World Event (in this case attack type)
target	A string	The name of the agent being targeted
duration	A long: > 0	How long the attack will be perpetrated
intensity	A double: 0.0 – 1.0	Level of intensity: translated to the degree at which a system is degraded.

Because of some simplifying assumptions we have made, these events are translated very simply into InfrastructureEvents according to Table 3. The current InfrastructureEvent types are given in org.cougaar.tools.csmart.Constants.InfEventType and are also listed in Table 3.

Table 3: CyberAttackEvents to Infrastructure Events

CyberAttackEvent Type	*InfrastructureEvent Type*
DOSNODE	NODE_BUSY
DOSNET	WIRE_BUSY
KILLNODE	NODE_DOWN
ISOLATENODE	WIRE_DOWN

Thus the ImpactModel for CyberAttackEvents only does this simple translation to produce Infrastructure events. The intensity and duration fields are copied directly without translation.

9.4.1.2 SimpleKEvent

SimpleKEvents are used to model kinetic events that do not move or vary in intensity over time. Examples include bomb blasts and earthquakes. A list of the current modeled Kinetic Events are given in org.cougaar.tools.csmart.Constants.RWEType and described in Table 4.

Table 4: SimpleKEvent Types

Constant	*Value*	*Kinetic Event Description*
FLOOD	"H2O flood"	A flood.
BOMB	"Bomb blast"	A Bomb
EARTHQUAKE	"Earthquake"	An earthquake.

The fields of a SimpleKEvent are nearly identical to those of the CyberAttackEvent and are given in Table 5.

Table 5: SimpleKEvent Fields

Field Name	*Possible Values*	*Description*
Type (from RealWorldEvent)	Constant from Table 4	The type of Real World Event (in this case kinetic event type)
location	A LatLonPoint	The location of the event.
Duration (T_{KE})	A long: > 0	How long the event lasts
Intensity (I_{KE})	A double: 0.0 – 1.0	Level of intensity of the event. This has special meaning to each event type.

The ImpactModel for SimpleKEvents is more complicated than that for CyberAttackEvents for two primary reasons:

- SimpleKEvents happen at a location on the globe and have radial impact. Thus they may affect a number of agents depending on their global location.

- SimpleKEvents have significant recovery times beyond the duration of the original event.

A general solution is given here, and no claims are made to the capability of this model to accurately represent these types of kinetic events.

Each type of SimpleKEvent is characterized by the variables in Table 6.

Table 6: SimpleKEvent Type-characterization Variables

Variable Name	*Possible Values*	*Description*
Max Radius (R_{max})	Positive Float (km)	The largest possible radius that an event of this type could have.
Max Recovery Time ($T_{max\ recovery}$)	Positive Long (ms)	The time that an agent at the epicenter will take to recover if the event is of maximum intensity
A node to net down ratio (α_{n2n})	Positive float	Used to determine whether a node or its network comes up first and at what ratio of total recovery time of an agent.

The ImpactModel will first determine which agents are affected by this kinetic event. A radius from epicenter (the lat-long of the SimpleKEvent) will be determined using the following formula:
Thus, the radius ($R_{effective}$) of the event is directly proportional to its intensity. Figure 16 illustrates the effect of intensity on event radius.

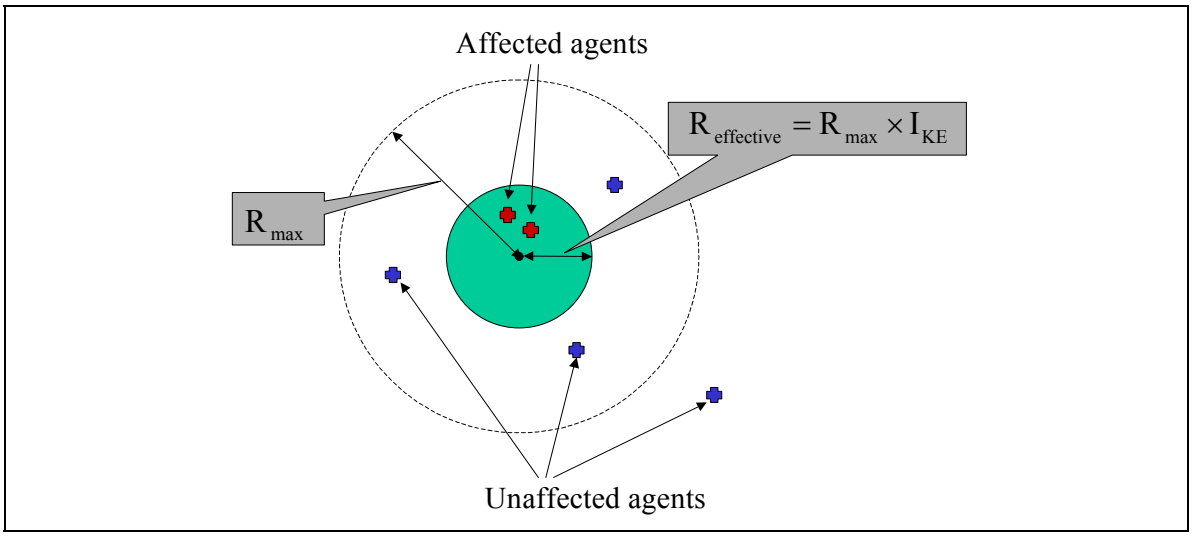


Figure 16: Intensity's Effect on Event Radius

Another assumption made by this model is that every agent affected by this event (that is, within $R_{effective}$) will be down for at least the duration of the event itself (T_{KE}). By "down", we mean that the both the network and node are down, not busy. In fact for this type of kinetic event, there is no concept of node or network busy. Kinetic events of this type either take things out-of-service or they don't. Therefore, kinetic events are only translated into `NODE_DOWN` or `WIRE_DOWN` `InfrastructureEvents`.

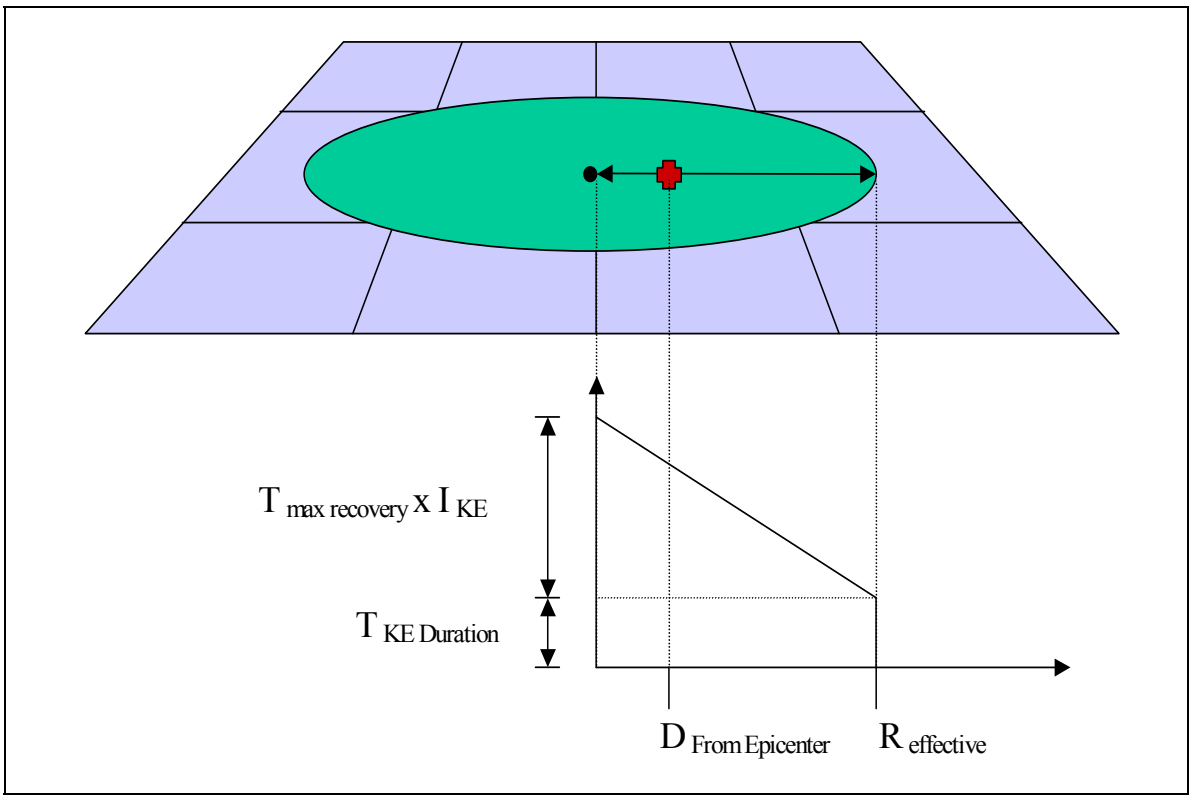


Figure 17: Agent Down time

Figure 17 represents the total down time an agent or network will experience due to a kinetic event and that time is calculated using the following formula:

$$T_{down} = T_{KE\ Duration} + \frac{T_{max\ recovery} \times I_{KE} \times (R_{effective} - D_{From\ Epicenter})}{R_{effective}}$$

As mentioned above, we are transducing SimpleKEvents into both `NODE_DOWN` and `WIRE_DOWN` `InfrastructureEvents`. So, what are the down times for each of these? That is determined by the node to net down ratio (α_{n2n}) mentioned in Table 6. If α_{n2n} is less than one, it means that the agent will recover before its respective network and the respective down times will be:

$$T_{node\ down} = T_{KE\ Duration} + \frac{\alpha_{n2n} \times T_{max\ recovery} \times I_{KE} \times (R_{effective} - D_{From\ Epicenter})}{R_{effective}}$$

$$T_{netdown} = T_{KE\ Duration} + \frac{T_{max\ recovery} \times I_{KE} \times (R_{effective} - D_{From\ Epicenter})}{R_{effective}}$$

If α_{n2n} is greater than one, it means that the network will recover before the node and the respective down times will be:

$$T_{nodedown} = T_{KE\ Duration} + \frac{T_{max\ recovery} \times I_{KE} \times (R_{effective} - D_{From\ Epicenter})}{R_{effective}}$$

$$T_{net\ down} = T_{KE\ Duration} + \frac{T_{max\ recovery} \times I_{KE} \times (R_{effective} - D_{From\ Epicenter})}{\alpha_{n2n} \times R_{effective}}$$

Figure 18 provides an example of a "down time" graph for a node-before-network recovery.

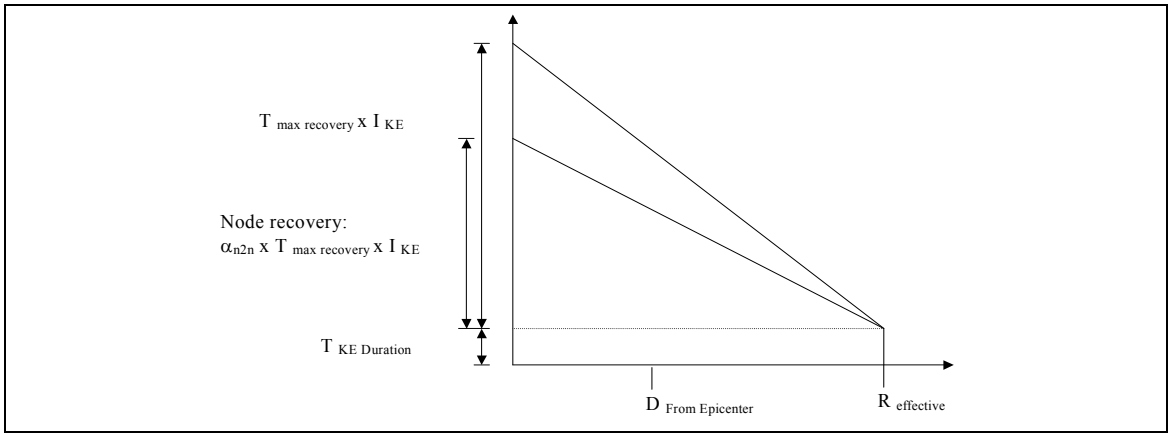


Figure 18: Node vs. Network recovery ($\alpha_{n2n} < 1$)

As you can see for each node affected by a kinetic event, two infrastructure events will be produced with different durations. One is a NODE_DOWN event, and the other is a WIRE_DOWN event. Both events have the same visibility time. That is, the agent and network will go down at the same time.

9.4.2 Creating new Event Types

When creating new Event types (either cyber or kinetic) the developer must keep a few things in mind.

- How will the producing component (PlugIn or script) describe the event? Does it have duration, intensity, or some other parameters? Does it have a target? How is that target described?
- How should the impact of the event be modeled? Does it have a radius? Does it affect just one agent? What kinds of infrastructure events are produced as a result of it?
- What state needs to be available from the transducer for the ImpactModel to correctly break the kinetic event into those events?

9.5 Injecting Real World Events into Cougaar

We have created a simple RealWorldEvent generation Plugin that reads a script and publishes the real world events described within it. The Plugin is capable of producing both CyberAttackEvents and SimpleKEvents.

9.5.1 Editing Real World Events file

The RealWorldEvents file is located in the appropriate "configs" directory and is titled (un-originally) RealWorldEvents.xml. The format of this file is expressed in an inline DTD (Document Type Definition), and the parser strictly enforces adherence to this DTD. When using CSMART to add an ABC Impact to your society, you will be prompted for the XML events file to use. CSMART will then handle putting this file in the necessary location for you. Figures 19 and 20 below present samples of the DTD and XML event specification, respectively.

```
<!ELEMENT EVENTLIST ((CYBER|KINETIC)*)>

<!ELEMENT CYBER (PARAM*)>
<!ATTLIST CYBER  TARGET CDATA #REQUIRED
           TIME CDATA #REQUIRED
           TYPE CDATA #REQUIRED>

<!ELEMENT KINETIC (PARAM*)>
<!ATTLIST KINETIC LATITUDE CDATA #REQUIRED
            LONGITUDE CDATA #REQUIRED
            TIME CDATA #REQUIRED
            TYPE CDATA #REQUIRED>

<!ELEMENT PARAM EMPTY>
<!ATTLIST PARAM NAME CDATA #REQUIRED
          VALUE CDATA #REQUIRED>
```

Figure 19: RealWorldEvent DTD

```
<EVENTLIST>
   <CYBER      TARGET="Victim"
               TIME="100"
               TYPE="DoS Node">
```

```
          <PARAM NAME="Intensity" VALUE="0.5"></PARAM>
          <PARAM NAME="Duration"  VALUE="3600"></PARAM>
    </CYBER>
    <KINETIC     LATITUDE="0.0"
                LONGITUDE="0.0"
                TIME="150"
                TYPE="Bomb blast">
          <PARAM NAME="Intensity" VALUE="0.1"></PARAM>
          <PARAM NAME="Duration"  VALUE="0"></PARAM>
    </KINETIC>
</EVENTLIST>
```

Figure 20: Example RealWorldEvent list

While the DTD and example may be enough to get started, some explanation should be made.

- Cyber and kinetic events can be mixed in the event list.
- The name of agents to attack in Cyber events must be the exact full name of the agent in your society.
- Look at the example agent ".ini" files in csmart/data/ul-test for usage details.
- There is no chronological ordering necessary. These events are all published at the beginning of simulation time, with appropriate visibility times.
- The TIME attribute will translate to the visibility time of the event and is given in milliseconds.
- The TYPE attribute must be a constant value from org.cougaar.tools.csmart.Constants.RWEType.
- The event generation PlugIn only knows of "Intensity" and "Duration" as possible parameters, because those are the only fields accepted by both CyberAttackEvents and SimpleKEvents. However, it should not cause problems if others are present.
- PARAM names are case sensitive along with the type values. When in doubt, use the above example as a guide.

Be careful when assigning intensities to Kinetic events. What may seem like a low intensity may actually be a much larger event. Table 7 gives the maximum recovery and radii of the three event types we are modeling. Keep in mind that these are worst case scenarios and in the case of bomb blasts – nuclear detonations.

Table 7: SimpleKEvent Constants

Type	*Maximum radius*	*Maximum recovery*	*Node-Net Recovery ratio*
H2O Flood	150 (km)	864000000 ms ≈ 10 days	0.5
Earthquake	50 (km)	432000000 ms ≈ 5 days	0.5
Bomb Blast	15 (km)	315360000000 ms ≈ 10 years	0.5

10. Future Development

The release of CSMART (v0.3) represented an initial prototype of our planned tool and the culmination of the ABC Testbed project. It focused on creating the foundation for an extensible tool suite for building, running, monitoring, and analyzing regular Cougaar societies and on performing experiments on those societies. At this stage of our development, we have created such a foundation with defined usage patterns and appropriate GUIs. In addition, we have re-implemented the functionality of the ABC Testbed and ALP Scalability Tool within this new CSMART framework.

Highlights of the CSMART capabilities include:

- Template-defined societies that can be easily tailored and executed for emulating large Cougaar societies
- Powerful visualization tools for analyzing Cougaar societies and understanding the behavior of these societies based on data collected from the distributed agent Blackboards (executed using CSMART or executed externally from CSMART)
- Metrics collection at the Node and Agent levels plus the ability to analyze this data using CSMART internal tools or external tools.

We plan to deliver the next release of CSMART (v1.0) on 31 October 2001 and to continue development through 2002. Prior to the v1.0 delivery, we plan a number of interim releases as we complete functionality that would be useful to the Cougaar community. Highlights of capabilities for our planned v1.0 release include:

- Support for defining a set of trials for an experiment by varying society parameters.
- Support for multiple topologies of agent connectivity
- Support for adding new PlugIns and for defining agents and other collections (e.g., an agent is a collection of PlugIns)
- Support for saving and loading collections of components in an XML or database schema
- Support for tailoring metrics collection for each experiment

- Integration of metrics collection with the Cougaar QoS Metrics Service (providing OS level metrics collection in addition to Node level metrics collection)
- Support for specifying and applying "impacts" (external events and operating conditions) for each experiment
- Support for visualizing agent societies and external impacts on geographic maps
- Support for managing configuration, experiment, and result information for repeating or adding to a set of experiment trials and for regression analysis across multiple experiments.

As mentioned above, we anticipate continuing the development of CSMART through 2002. Highlights of capabilities for our planned v2.0 release (October 2002) include:

- Graphical specification and visualization of society construction
- Constraint checking or closure analysis for society construction
- Society construction help provided by a wizard-like capability
- Generic PlugIns (expanders, allocators, and aggregators) tailored through XML rule files
- Topology variation across multiple trials
- Automated mapping of Nodes to Hosts, including variations across multiple trials
- Enhanced "external impacts" including DDOS attacks and location/time-course kinetic events
- Animated playback of Society/agent processing including the overlay of time-varying metrics on agent topology maps
- Scalable visualization techniques for societies comprising 100s of agents and 10,000 plus graph nodes
- Scalable information retrieval for societies comprising 100s of agents
- Graph comparison and graph filtering techniques.

CSMART has proved to be a valuable tool for the exploration and understanding of agent societies based on the Cougaar agent architecture. While our tools have been exclusively focused on interacting with and interpreting data from Cougaar agent software components, we believe that the experimentation techniques, visualization techniques and graphic user interfaces can be extended to large-scale distributed, composed systems in general. To realize this generalization, CSMART would need data collection and appropriate formatting of this data particular to each new system of interest. In addition, one may wish to specify particular metrics for a new system. Then, the full power and utility of CSMART can be used to gain insight into the behavior of new distributed and composed systems.

11. References

CSMART User's Guide, v0.3, July 2001, www.cougaar.org.

Cougaar Architecture Document, v8.2, July 2001, www.cougaar.org.

Helsinger, A., Ferguson, W., Lazarus, R., "Exploring Large-Scale, Distributed System Behavior with a Focus on Information Assurance," Proceedings of the second DARPA Information Survivability Conference and Exposition (DISCEX II), Anaheim, CA, June 2001.

"Modeling the Effects of Events in Cougaar/Ultra*Log Societies," ABC Project report, May 2001.

"Measures of Success," ABC Project report, August 2000.